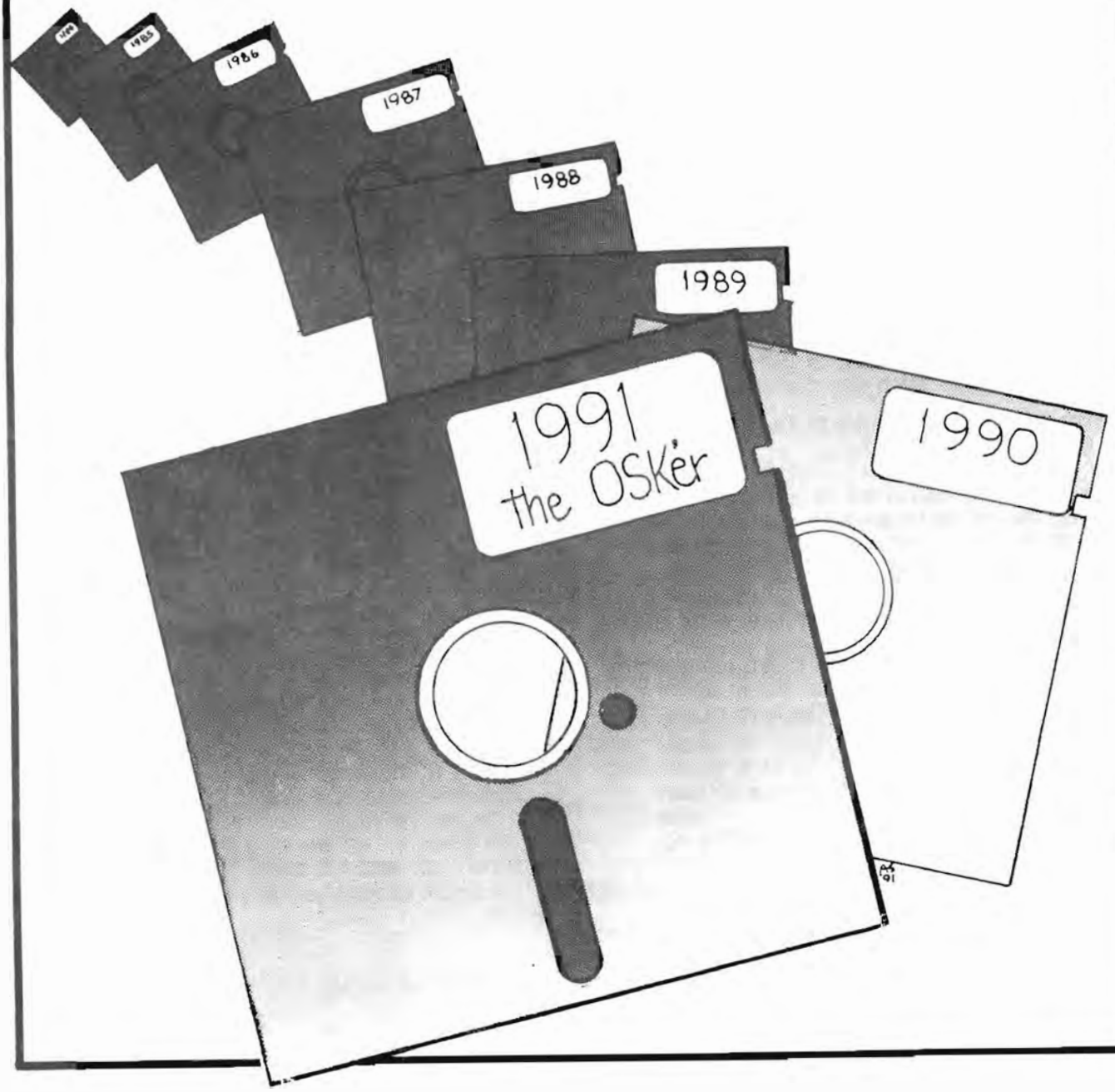
$2.00

THE OSK'ER®

News and Views in the World of OS9/68000 and 6809

ACCESSING THE NEW YEAR!



```
            Directory of /dd/SYSOP/OSKer/I4 11:13:57
 Owner  Last modified Attributes Sector Bytecount Name
------- ------------ ---------- ------ --------- ----
Levinsn 90/10/12 0709  ----r-wr      1      8156 Basic09_Intro
StG     91/04/23 1104  ----r-wr      4      8856 dlink.c
Editor  91/04/21 2337  ----r-wr      8     13196 Editor_Rambles_On
Fleagle 91/12/07 1324  ----r-wr     11     12780 O_Say_Can_You_C
Pollock 91/10/13 0210  ----r-wr     14     10640 System_Calls
Editor  91/04/23 2313  ----r-wr     16       707 Z_End
```

CONSIDERATIONS FOR SUBMITTORS

There are a few things ya'll can do to make my life easier. When sending in submissions of any type (text, program), please make sure of a few things first:

1) The file(s) are in ASCII text, preferably without linefeeds.
2) Do not justify the text, our processing will align it for you.
3) Do NOT indent paragraphs, but do leave a blank line between them.
4) Please don't leave any left margins or hyphenate any words.
5) Be sure to include your name and how to contact you.
6) Best way to send it is via a network: Sysop@Root on StG-Net, or 72427,335 on CIS.
7) If sending by mail, use OS9 CoCo Format, 5.25" or 3.5" (720k), or ST format 3.5" (720k), or MM1 format 3.5" (1.44M)
8) If you want to archive it, please use the OS9 .AR format.

the OSKer

'the OSKer' is printed monthly by StG Computers inc, P.O. Box 24285, Speedway, Indiana, 46224. The president and editor is Scott Griepentrog, V.P. Jim Hutchins, Secretary Chris Swinefurth, and Treasurer Dave Henk.

Subscriptions to the OSKer are $12 within the U.S., $15 for Canada, and $20 overseas.

Editing and Layout for the OSKer is done completely under OSK, using a prototype MM1. uMacs is used for pre-editing of text, and a custom routine is used for formatting and layout. An ALPS Allegro 500 (flat paper path, 24 pin) printer is used to produce the initial copy for duplication by offset, as well as printing the mailing addresses on the back cover. The subscription list is also kept in a database under OSK.

To prevent a conflict of interest, StG Computer inc., as both publisher of the OSKer and having ownership of software, will not directly advertise in this magazine, nor will the editor in any way promote said software.

The Cover artwork is done by Alan Sheltra

Introduction to BASIC09
by Eric Levinson 9/9/90

Many people that have used the standard BASIC from the Disk Extended Color Basic believe that it is the best BASIC they have seen, or the best in its time. This is true, the Disk Extended Color Basic (DECB) is a very powerful programming language allowing certain features like user-defined functions, direct buffer manipulation for graphics and fast execution. In addition many companies have written compilers to further compress the BASIC source code into an intermediate pre-interpreted executable code.

A few things that DECB lacks, direct communication with the operating system, higher structure (more on this), and parameter passing. In addition OS9 offers the user the ability to run more than one program at a time.

Why BASIC09?

BASIC09 was developed and structured after PASCAL. BASIC09 is a highly structured programming language that affords speed, versatility, direct access to the DOS, modularity, parameter passing and encapsulation.

BASIC09 recognizes four data types. INTEGER, REAL, STRING and BOOLEAN. In addition complex data types can be defined with the TYPE command (any of the above can be mixed into one type). Up until now, the DECB users have only used the REAL and STRING type. Computers cannot directly manipulate real numbers because an element of indescreteness occurs between two real numbers. If I asked you, "How many numbers are between the real number 0. and 1.?" you would have to say that there are an infinite numbers between those numbers. You could have 0.5, 0.25, 0.26, or even 0.00000000000000000001 for that matter. A REAL type variable is considered "indiscrete" because there is no logical step between numbers. In DECB, a REAL number consists of a 5 byte coded string. See MKN and CVN commands in your DECB manual for more. When DECB does arithmetic, it has to manipulate all 5 bytes. This can take alot of time to do many multiplies, or exponentiation, especially if they are in a GOTO loop and are repeated over and over again. This is what the INTEGER type is for in BASIC09. If you are doing simple arithmetic, the INTEGER type consists of 2 bytes, but these bytes are not coded. They are the raw data in binary representation. The INTEGER variable type can go from -32768 to 0 to +32767, modulo, which means that +32767 + 1 = -32768. When you assign a variable to be type INTEGER in BASIC09 it gets acted on almost 8 times faster than a REAL type number in DECB. So you can see, for its purpose, INTEGER can be a real time saver. The INTEGER type is a "discrete" data type because each number in the sequence has one number before it, and one after it. There are no numbers between 0 and 1 in the INTEGER data type. The STRING data type is used so you can set up strings to hold data. If a $ is used after a variable name, it automatically gets 35 bytes of string space. Last but not least is an "enumerated" data type called BOOLEAN. It is discrete, and has only two values, True and False. It is called "enumerated" because the values are not string results, not are they numerical results. Any time two variables of the same type are compared with =, <, >, <>, <= or >=, the result is said to be BOOLEAN. The result is either True, or False. There will be example programs to follow.

Structured Languages

For almost a decade Computer Scientists argued that the GOTO statement was not required, and not needed, if the correct statements to replace them were available. These looping statements are as follows: REPEAT/UNTIL, WHILE/ENDWHILE, LOOP/ENDLOOP and FOR/TO/NEXT. All of these structured statements allows the program not to consist of "spaghetti" code as found formally in old versions of BASIC, like DECB. In addition, programs can be written WITHOUT the use of line numbers. This also reduced the clutter that occured, if GOSUB was required, then a line number can be entered on the line to be the subroutine.

Type in the following BASIC09 program. To start BASIC09, place your OS9 disk in your drive 0. Type DOS, and wait for the OS9: prompt. Once in OS9, insert your BASIC09 disk and type: BASIC09.

After a few seconds of disk churning, the following message will appear:

```
            BASIC09
      RS VERSION 01.00.0X
COPYRIGHT 1980 BY MOTOROLA INC.
  AND MICROWARE SYSTEMS CORP.
   REPRODUCED UNDER LICENSE
       TO TANDY CORP.
    ALL RIGHTS RESERVED.

Basic09
Ready
B:
```

You are now ready to learn the three modes of BASIC09.

The first mode is the command mode (which is where you are right now.) You may type LOAD, SAVE, RENAME, LIST, E,

KILL, DIR, MEM and a few other commands here. To start typing a program in, choose a name like TEST and type E TEST at the B: prompt. The E command tells BASIC09 to go from the command state to the editor.

Now it will show:

```
PROCEDURE test
*
E:
```

The * means that the editor pointer is at the top of the file. Type the following lines, making sure you enter a space before the line. The space tells the editor you wish to insert a line. The BASIC09 editor works exactly like the standard EDIT command works in OS9. Some basic commands while in the editor:

```
D        Deletes the current line
S/text/  Searches for the first occurrence of text
S*/text/ Searches for all occurrences of text
-*       Places the pointer back to the beginning of the BASIC09
         buffer
+*       Goes to the end of the buffer
```

Type the following as it appears here:

```
 DIM x,y:INTEGER
 DIM a:STRING[80]
 DIM r:REAL
```

```
DIM b:BOOLEAN

LOOP
INPUT "Enter a number ";x
INPUT "Enter another number ";y
PRINT "X * Y = ";x*y
INPUT "Enter a decimal number ";r
PRINT "X * R = ";x*r
INPUT "Enter a string ";a
PRINT "You have entered: ";a
b := (x = y)
PRINT "D
INPUT "Do you want to try again? ";a
EXITIF a="NO" or a="
ENDLOOP
END
```

After you have entered this code in, type Q on the line by itself.

While at the B: prompt, you may type LIST to see your program. If any errors were reported, you will see the message ERR before each suspected line.

BASIC09 will now attempt to write it into memory so it may be executed. If you get any errors, BASIC09 will show the hexidecimal memory location errors where the error occurred and you may go back into the editor and fix it.

Notice, no line numbers? That is because we put the LOOP in the program. Without the LOOP command, we would need to use at least one line number at the beginning and have it go back to the beginning to start over.

Everything is pretty much self explanatory, except the b := (x = y). We are asking the BASIC09 interpreter to compare X and Y. If they are equal, store a True in B. If they are not, store a False in B. Notice after you entered that line if you went up a line, the parentheses would be gone, like you never entered them. That is because BASIC09 knows where they are required, and removes them when they are not.

Since BASIC09 remembers the name of the procedure, if you need to go back, simply type E. If you want to save the procedure simply type SAVE at the B: prompt. To run the procedure, type RUN.

The third mode of BASIC09 is the DEBUG mode. You can enter this mode two ways. One way is by placing the PAUSE command in your BASIC09 program. The other way is by pressing CNTRL C while your program is executing. While in DEBUG you will see a D: prompt. You can issue BASIC09 commands like: PRINT a or CONT to continue. Press Q to get out of the DEBUG mode. While in the B: and D: modes, you may type $ to run a shell. Type ex in the shell to return to where you were in BASIC09. If you just want to get a directory of a disk and don't want to leave BASIC09, simply type $dir at the B: or D: prompts.

When done, simply type BYE. This will take you out of BASIC09 and back to the OS9: prompt. MAKE SURE YOU SAVE YOUR FILE FIRST! when you type BYE, there is no second chance.

My next article will be a series of features of BASIC09.

If you have questions, feel free to write me directly. My address is:

Color Galaxy Inc.
Eric Levinson
24415 Marquis Ct.
Laguna Hills, CA 92653

StG Net Software/Login Pakg V3.0

Complete software to run your own BBS!
For OS9 Level II /Coco 3 (OSK Version soon!)
Extremely flexible!! ... Includes:

- Run a powerful, multi-line BBS without losing the use of your computer! Run up to 8 lines.
- Hook up with an international Network, the StG Net.
- Net Account included for your system so you can "net" with other StG Systems imediately. Other Network interfaces coming soon.
- Complete E-Mail and Net-Mail Message System. Binary or Text files can be sent as private mail.
- Flexible Menu system allows you to create your own menus in ANSI or OS9 Graphics. Almost ANY OS9 program can be run from a menu (Std I/O). Sample Menus included, so you can go right on-line.
- DES (Data Encryption Standard) Password Protection
- System utilities include: Mail, News, Chat, TSmon, Login, Help, Netxfr, Option, Status... and many more...
- Help Utility included, gives you an On-line manual. Also includes printed insallation manual.
- Xmodem/YModem/Kermit Protocols for file transfer
- Includes FREE upgrades to Version 4.0 (Coming soon) All valid systems will receive upgrades via the net!
- Includes AniMajik's Games and Utilities Pak, made specially for the StG Net System.

AS0011 (StG Net Login Pkg + CDI + IRQ Fix $49.95
(Please include $3.00 S&H)

STARDUMP

A Full Color VEF Format Picture Dump Utility for the NX1000 Rainbow Printer. Sample VEFs Pictures included.

AS0014 (Includes Disks and Printed Manual) ..... $ 19.95
(Please Include $3.00 S&H)
AS0014M (Save $5.00) Download program and doc via modem supplied as an ARed File (NO S&H Required!) ..... $ 14.95

Coco Tycoon - by AniMajik Productions

Create a "Monopoly" on your Coco3 (OS9 L2 Req)
1 to 4 Players Even play against the Coco...
Plays just like the Board Game...
(Available mid=April)

(Reg. Price $19.95)
Special Intro Price **..... $14.95**
(Add $3.00 S&H)

(Take $3.00 off for Downloading via Modem/ S&H not Required!)

AniMajik PRODUCTIONS

Send Checks or M.O.'s
P.O. Box 38713
Hollywood, Ca. 90038
(818) 761-4135 (Voice)
(213) 460-2968 (FAX)

MONSTEROUSLY GREAT DEALS...
AT MORE THAN HUMAN PRICES!!!

(Prices Subject to Change without Notice)
(Ca. Residents please add 6% sales tax)

TSHELL - by Paul Pollock

(For your Coco 3, OS9 Level 2 System)

A Revolutionary New Program..."TShell" does most of what Multi-Vue does at many times the speed! TShell will run most programs with one keypress...and will use standard MV AIF files. Delete, Copy, Rename files all with one or two keypresses!

Many Utilities Included * WINDINT and MV NOT Req!**

AS0012 (Includes Disks and Printed Manual and accessories)
(Please Include $3.00 S&H) $39.95
AS0012M (Save $13.00!) Download program Direct by terminal (From any StG Node, call for more info) Includes everything as above. Manual is in pre-formatted docfile ready for your printer. (No S&H Needed!) $29.95

Coming Soon!

*** DB9 -The Best Database for OS9!**
(Call or write for details and availability)

*** RECLAIM - The Disk Doctor**
"Reclaims" deleted files, fixes bitmap and sector problems on your floppy or hard drives.
(Call or write for details and availability)

*** MemMatch! - Coco Concentration...**
(512k and OS9 Level II Req.)

HARDWARE

3 Button Serial Mouse - Perfect for your New MM/1 or TC9! (works with IBM compatibles too!)

- 3 Button Opto-Mechanical Serial Mouse
- Smooth, aestetically appealing, ergonomic design
- High Precision 250-1000 DPI
- Precision "Click-Style" Buttons
- Includes user guide.

SPECIAL PRICE!

AH0023 (Include $3.00 S&H) ..... $ 29.95

Mouse Pad (Blue or Grey)
AH0024 (Please include $1.50 S&H) ..... $ 5.95

"Mouse House" - Mouse Holder

Holds your mouse - attaches to CPU or Monitor

AH0025 (Include $1.50 S&H) ..... $ 3.99

Mouse Kit - Includes:

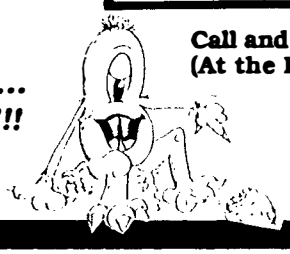
- 3 Button Mouse
- Mouse Pad
- "Mouse House"

AH0026 (Include $3.00 S&H) ..... $ 34.95

Hard Drives (SCSI)

Quantum SCSI Hard Drives (3 1/2" half ht- will fit inside your MM/1)
FAST! 18ms Drives
64K Memory Cache (Call for more info)

NEW Low Prices!
#AH0028 - 80 Meg Quantum SCSI HD .....$ 399.95
(Please include $7.00 S&H with each HD order)



Call and Browse our Catalog at any of these BBSes
(At the LOGIN prompt, type "an!majik")

(818) 761-4721 (MODEM)
(818) 772-8890 (MODEM)
(403) 329-6438 (MODEM)
(904) 595-2184 (MODEM)

Dlink.c

```
/*
 * dlink/move - utilities for OS9 and OSK
 *
 * PD 1991 by StG
 *
 * to compile: cc dlink.c
 * then      : chd /dd/cmds
 *             dlink dlink move
 *
 * Instructions: Basically, DLINK <from> <to> -or- MOVE <from> <to>,
 *               where <from> is a directory (read warning) or file
 *               that already exists, and <to> is a file that you want
 *               to exist.  Both commands will create the <to> file
 *               and link it to be the same as <from>.  The MOVE command
 *               then deletes the <from> file so that the original is
 *               basically moved.
 *
 * How it works: OS9 already supports linked files, to a degree.  There
 *               is a link count in the FD (file descriptor) sector for
 *               each file.  If more than one directory entry are
 *               pointing to the same FD, both will be able to access it.
 *               In this case, the link count would be two, indicating to
 *               the OS9 delete command that it should not actually
 *               remove the file when only one of the links has been
 *               deleted.  Instead, it subtracts one from the link count.
 *
 * Problems: The OS9 dcheck command does not understand links, and
 *           therefore consideres them to be an error.  The OSK dcheck
 *           command is ok.
 *
 * Warning: Linking to a directory will likely cause it's '..' pointer to
 *          be incorrect.  This will cause programs that use '..' in
 *          making a scan of the directory structures to muss up.  One
 *          good example of this is the pwd (OSK: pd) command.  It will
 *          report the wrong directory (if it doesn't error out) when a
 *          '..' pointer is bad.
 *
 * Short Form: Link to directories only at your own risk!
 *
 * NOTE: you can rename this to LINK iff you don't use OS9's link command
 *
*/

#define ERR (-1)
#include <stdio.h>

extern int errno;       /* OS9 error code */

int ff,fd,tf,td;        /* from/to file/dir */
int af;                 /* device# file */
long dir=0;             /* directory flag & lsn storage */
long fre=0;             /* free space in td */
char buf[256];          /* shared buffer */
char foo[256];          /* if argv[2] has to be foo'd with */

struct                  /* directory structure */
{
   char nam[28];        /* file name */
   long lsn;            /* pointer */
} dir;

long lseek();           /* make sure we know it is long */

/* returns path to file */
char *
path(s)
char *s;
{
    char *b=buf;
    char *p=s;

    /* insure there is at least one '/' */
    while (*s) if (*s++=='/') break;
    if (!*s)
    {
        /* return '.' for current directory */
        *b++='.';
        *b=0;
        return(buf);
    }

    /* skip to end and back up to last '/' */
    while (*s) s++;
    while (*s!='/') s--;

    /* copy into buf & return */
    while (p<s) *b++=*p++;
    *b=0;
    return(buf);
}

/* returns just name of file */
char *
file(s)
char *s;
{
    char *b=buf;
    char *p=s;

    /* insure there is at least one '/' */
    while (*s) if (*s++=='/') break;
    if (!*s)
    {
        /* return whole name */
        while (*p) *b++=*p++;
        *b=0;
        return(buf);
    }

    /* skip to end and back up to last '/' */
    while (*s) s++;
    while (*s!='/') s--;

    /* copy into buf & return */
    s++;
    while (*s) *b++=*s++;
    *b=0;
    return(buf);
}
```

```
/* quick case insensitive string compare */
same(s1,s2)
char *s1,*s2;
{
    while (*s1 && *s2) if (tolower(*s1)!=tolower(*s2)) return(0);
    else
    {
        s1++;
        s2++;
    }

    if (*s1 || *s2) return(0);
    return(1);
}

/* set bit 7 on name string for dir entry */
set7(s)
char *s;
{
    while (*s) s++;
    *--s|=128;
}

main(argc,argv)
int argc;
char **argv;
{
    if (argc!=3)
    {
        printf("use: %s (from) (to)\n",*argv);
        printf("  %ss 'from' dir or file to 'to' dir or file'\n",*argv);
        exit(0);
    }

    /* open from file */
    ff=open(argv[1],3);
    if (ff==ERR)
    {
        /* try to open as dir? */
        ff=open(argv[1],128+3);
        if (ff==ERR)
        {
            printf("%s: can't open %s\n",*argv,*++argv);
            exit(errno);
        }
        dir++;    /* set flag - we are moving a directory! */
    }

    /* check to path to see if a directory */
    if (access(argv[2],128+1)!=ERR)
    {
        /* we need to fudge argv[2]
        /* user has supplied directory to link/move to but not file
        /* take file from argv[1] and tack on argv[2]
        */
        strcpy(foo,argv[2]);
        strcat(foo,"/");
        strcat(foo,file(argv[1]));
        argv[2]=foo;
    }

    /* open from and to directories */
    fd=open(path(argv[1]),128+3);
    if (fd==ERR)
    {
        printf("%s: can't open %s\n",*argv,buf);
        exit(errno);
    }

    td=open(path(argv[2]),128+3);
    if (td==ERR)
    {
        printf("%s: can't open %s\n",*argv,buf);
        exit(errno);
    }

    /* both paths must be on same device! */
    _gs_devn(fd,buf);    /* get entry, */
    strhcpy(buf,buf);    /* fix 7bit high on last char */

    _gs_devn(td,buf+32);    /* sneak room in buf */
    strhcpy(buf+32,buf+32);

    if (strcmp(buf,buf+32))
    {
        printf("%s: can't operate between different devices\n",*argv);
        exit(1);
    }

    /* open path direct to device */
    *buf='/';
    strcpy(buf+1,buf+32);
    strcat(buf,"@");
    af=open(buf,3);
    if (af==ERR)
    {
        printf("%s: can't open to device %s\n",*argv,buf);
        exit(1);
    }

    /* search to directory in case file is already there
    /* and for a free spot to put it in
    */
    file(argv[2]);    /* put name of file in buf */
again:
    fre=0;
    lseek(td,0L,0);

    /* if we are link/moving a dir, grab the '.' reference from
    /* where it's going ... */
    if (dir)
    {
        read(td,&dur,32);
        read(td,&dur,32);
        dir=dur.lsn;
    }

    while (read(td,&dur,32)==32)
    {
        strhcpy(dur.nam,dur.nam);
```

```
        if (same(dur.nam,buf))
        {
            printf("%s: file %s already exists in ",*argv,buf);
            printf("%s\n",path(argv[2]));
            exit(1);
        }
        if (!*dur.nam && !fre) fre=lseek(td,Ø1,1)-32;    /* free spot */
    }
    if (fre)    /* seek back to empty entry */
    {
        lseek(td,fre,Ø);
        read(td,&dur,32);    /* lock it */
        if (*dur.nam) goto again; /* oops, somebody else got there first *

        lseek(td,fre,Ø);
    }

    /* path td is now ready to receive new entry
    /* next go through fd to find lsn to link to
    */

    file(argv[1]);    /* put <from> filename in buf */
    while (read(fd,&dur,32)==32)
    {
        strhcpy(dur.nam,dur.nam);
        if (same(dur.nam,buf)) break;
    }
    /* check once more in case loop at end! */
    if (!same(dur.nam,buf))
    {
        printf("%s: oops, can't find file %s in ",*argv,buf);
        printf("%s\n",path(argv[1]));
        exit(1);
    }

    /* LSN we want is in dur.lsn, we can put new name in and
    /* write to td.  But, first set FD for extra link count
    /* in case program gets blown away.  Better an extra link
    /* than one short!  Exception: running as move
    */

    if (tolower(**argv)=='m') goto nolink;  /* program is named move */

    /* re-use fre var as a LSN pointer */
    fre=dur.lsn;

$ifdef OSK
    /* OOPS! OSK V2.3 now allows VARIABLE SECTOR SIZE, which means
    /* that we might need to modify fre to get to the right sector!
    */

    /* read lsn Ø into buf */
    if (lseek(af,ØL,Ø)==ERR) exit(errno);
    if (read(af,buf,256)==ERR) exit(errno);

    /* the two bytes at offset Øx53 are the sector size */
    if (*(buf+Øx54)) exit(-1);    /* should always be zero */
    if (!*(buf+Øx53)) goto skip; /* whew, this disk doesn't do that */
    /* multiply by number of multiple of 256 of sector size */
    fre*=*(buf+Øx53);

skip:
    fre*=256;

$endif

    /* seek to FD and read */
    if (lseek(af,fre,Ø)==ERR) exit(errno);
    if (read(af,buf,256)==ERR) exit(errno);

    /* increment link count */
    if (*(buf+8)<Ø)
    {
        printf("%s: too many links to %s\n",*argv,*++argv);
        exit(1);
    }
    (*(buf+8))++;

    /* before writing FD we must close our path to file
    /* we are done with it anyways, were only keeping it
    /* open to prevent delete while in use
    /* during close OS9 re-writes FD (would wipe out change)
    */
    close(ff);

    /* write FD back */
    if (lseek(af,fre,Ø)==ERR) exit(errno);
    if (write(af,buf,256)==ERR) exit(errno);

nolink:

    /* now we can write out the new directory entry */
    strcpy(dur.nam,file(argv[2]));
    set7(dur.nam);
    if (write(td,&dur,32)==ERR) exit(errno);

    /* link has now been accomplished
    /* time to check out .. entry (if dir) and handle move
    */

    if (dir)    /* original file to move was a dir? */
    {
        /* open new file and check .. pointer */
        tf=open(argv[2],128+3);
        if (tf==ERR)
        {
            printf("%s: UhOh! can't open %s\n",*argv,argv[2]);
            exit(errno);
        }
        read(tf,&dur,32);
        if (dur.lsn!=dir)    /* dot dot is wrong */
        {
            if (tolower(**argv)=='m') /* we are move! */
            {
                printf("Updating %s/..\n",argv[2]);
                lseek(tf,ØL,Ø);
                dur.lsn=dir;
                write(tf,&dur,32);
            }
            else printf("Warning: %s/.. is wrong!\n",argv[2]);
        }
```

```
        close(tf);
    }

    /* if we are move, unlink (delete) the original file */
    if (tolower(**argv)=='m')
    {
        /* note: fd still has original entry locked, back up & wipe out */
        lseek(fd,-32L,1);
        *dur.nam=0;
        dur.lsn=0;
        write(fd,&dur,32);
    }

    /* close our myriad of files */
    close(af);
    close(td);
    close(fd);
}


#ifndef OSK
/* this section included for OS9 which doesn't have _gs_devn call */

#include <os9.h>

_gs_devn(pn,buf)
int pn;
char *buf;
{
    struct registers r;

    r.rg_a=pn;
    r.rg_b=SS_DEVNM;
    r.rg_x=buf;
    return(_os9(I_GETSTT,&r));
}

#endif
```

Editor_Rambles_On

And on, and on, and on...

In the previous issue, (supposedly) dated September of last year, I started this regular column with the complaint 'How did I ever convince myself this would be an easy job'. I was thinking about how it seemed to take two months to put together each magazine. At the time I was in the middle of several rush projects, and I had only with that issue settled on a simple process for editing, formating, printing, addressing, and mailing the magazine. I have learned a lot about the trials of publishing from this, and I must be a glutton for punishment because I am not about to quit.

I have received many calls from concerened subscribers during the period we have been 'off the air'. I apologize sincerely to all of those who had faith in this project and whom I have disasppointed. But now that StG inc. has completed it's move and is once again making enough spare money to handle the nearly thousand dollars it takes to put out an issue (that's *after* subscriptions, advertisements, and start-up costs!), let me assure you all that we will continue to put out. When I started this magazine, I made the subscription rates low on purpose - I hate to pay a lot for magazines myself. If I had wanted the OSKer actually make money, the rates would have to be at least double. I have considered raising the rates even though, but I will put that off indefinately. If we can bring in some additional regular advertising, it shouldn't be necessary to charge any more per issue. Of course, I've shot a hole in that plan already - advertisers want to have a reliable publication as well as the readers do. As my Mom & Dad love to remind me, it takes a lot of hard work to make it on your own. Yes, Mother, I have my work cut out for me.

Of course I'm not the only one working on this (not that anyone else is to blame though). My thanks go to Alan Sheltra for his wonderful support (and wonderful artwork - just wait till you see next month's cover!!) and my friends (Hi Bug!) for getting out of my hair now and then to let me get it done. And special thanks go to those who have written articles, suggested improvements, and offered help!!!

Okay, now it's time to get down and dirty. I don't have any letters to the editor for today, but I do have a few personal opinions to vent.

WARNING: the following is highly flammable - read only in a well ventilated area!

It would seem that the whole OS9 community is having problems. We've got magazines failing (or at least slow) right and left, software houses folding, and certain computer companies that keep promising the new machines but (it would seem) never deliver on time. I ask you, WHAT THE H*** IS GOING ON HERE? Why does it seem as though the OS9 community (except the industrial people) seem to be abandoning the long held commitment to the best in Multi-User Operating Systems? Is this a real problem, the beginning of the end, or just a phase? Just what is going on, anyways?

Well, I have a few answers. But time will only tell all for certain.

In the meantime let me tell you a story or two.

In the Indianapolis area there is a group of OS9 enthusaists that grew out of the local Color Computer Club. We get together now and then and help each other with equipment or software problems, discuss new things we have discovered, and talk about where things going. Up until recently, we all had OS9 or OSK machines and never even considered much else. But in the last few months, a number of our already few have 'sold out' and bought 386 machines.

Now some would say that they should be lynched for giving up the fight for the Better Operating System. Some others would excumunicate them from the group for giving into the 'PC' world's single tasking environment. But I still accept them. I even help them with their machines. Because I agree with their decision.

Now before ya'll start getting any ideas about tp'ing my house, stop for a second and hear me out. These fellows, who shall remain nameless (as the innocent should), were waiting for the 'fabled' CoCo4 machine.

That right, the do-all 680x0 machine that would solve all our problems and limitations OS9 on the CoCo. They waited faithfully for years for it to come out. They were encouraged by the announcements of two such machines, put off upgrade plans, and saved up their money. They enjoyed themselves for a while arguing over which one was better. They warmed up their checkbooks as they anxiously counted the days until they could actually have one of their very own. Some of them even sent in their money in advance.

But that day came and went. No machine. And another day came and went, again with no hardware to put on their desk. No fast machine in a neat case with a mouse next to the keyboard and windows to peek through. No new manuals to go through, no new software to play with.

So they came to the conclusion that it just wasn't going to be. That these new machines would never come out, or never have enough software to run on them. Or never be any better than the PC world with it's super fast machines and hi-res windows. There were tired of being left out in the cold with their poor little 8-bit (bus) machines. That money in their pocket kept itching every time they drove past a new store selling PC's. So finally they couldn't stand it any longer. They bought PC's.

Now they are busy playing with thier new toys, new software, new mice, new windows, and having a grand time. Do they regret their decision? Well, not much. Do they still think that OS9 is the best operating system? Well, it still is best at multiple tasks, but it doesn't have as many programs for it. Do they miss being able to run multiple programs? Well, not really. They have Microsoft Windows version 3 that allows them to at least flip between multiple programs if not let them process at the same time very well.

You see, this is going on througout the OS9 community. People are leaving OS9 for MSDOS just because it's there. And also because MSDOS is finally catching up Windows-wise. And most of all, because people have promised various Machines, Software, and Magazines and not delivered on time. I am not trying to blame any particular individual. As I have pointed out, I am also to blame for not being on the ball.

Oh sure, you think, these guys (and others) that we are loosing to PC's will come back once we get going right? Fat chance, considering what they paid for their 386 machines and the software to run them. These guys put more money into their new machines than what they would have for a TC70 or MM1 (or similar machine) fully decked out. They did so because the PC's were available right then. I think these people making

the 'CoCo4' machines (you know who you are) made a big mistake by not getting them out by Christmas. But then, I made a big mistake by not getting an issue out by Chrismas..

We need the machines and the software NOW, not in a month or two. Not in a year or two. If we are going to keep the dwindling numbers of OS9 enthusiasts from near extiction we all need to get moving and start cranking out top notch equipment and machines right away. We need the same kind of ingenuity that has blown the MSDOS world away before to come up with new ideas that will grab people's attention. Show them what can be done if you do it right.

We need people to get the job done, not promise something and apologize when they can't make due. I've learned my lesson - the OSKer will be coming out regularly from now on. The question is, have the rest of us?

Time for another story.

A friend of mine was showing OS9 on a CoCo to a PC programmer the other day. He was immediately impressed, and remarked how that was the kind of system he wanted to have. But could he get it for his PC? I told him about OS-9000, and how it requires a 386 (at least) and isn't as fast and effiecient as plain OS9 on a Motorola Processor. He said "Oh" in a way that meant, "Well, if it doesn't work as well on a PC than it must not be so hot after all". I explained to him some of the differences between Motorola 68000 processors and the Intel 80x86 ones. In the end I discovered that he didn't really know much about Motorola processors, and had never heard of OS9. This actually didn't suprise me, as less than one person in ten that I talk to has.

There is a bias in the overall community of computer people out there - but one not due to anything IBM or Intel has done, but rather due to what Motorola and Microsoft _haven't_. There has been little effort on the part of Motorola (who's processors are used more than Intel's in Japanese products) and Microware (who has a large industrial market in Japan too) to educate us here in the States about their products. The Japanese love to find the simplist, easiest to use tools to make their products. They should know what their doing, considering the amount of robotics they use. And they are certainly making enough money at it - they keep buying pieces of us at prices higher than they might have to.

If we are to survive at all, we must grow. We must attract people to our 'camp' faster than we loose them. I don't believe it is any secret that this is not the case right now. But we are on the edge right now; either we fall off that edge into oblivion or we pull back in the nick of time. We have been given a second chance with these new machines. One more chance to get it right.

The OS9 community has been brusied over and over by well-intentioned people who really meant to make good on their promised but for one reason or other were not able too. Of course, there have been those few who were just outright crooks, too. But for the most part we have had decent people come up with new products and bring them out to the market, if a little late than planned on, and end up benefiting the whole OS9 community. And if we keep it up, there should be OS9 enthusiasts still hacking on into the next century.

But how does this differ from the so called 'real' market - those companies who develop and market PC products? They have their share of problems too. Big name companies that take forever to get a new machine or software version out. Code that has enough holes in it to be mistaken for swiss cheese. The people who deal with PC machines on a regular basis (especially when it comes to newly developed stuff) have become accustomed to having problems. How long ago was it that OS/2 was announced? It's only been just recently that version 1.3 has been released - all the previous versions have been full of bugs, take gobs of memory, and are slow. Guess what! This new version takes over 20 disks to install (that's 1.2 meg disks!) and runs halfway decent on a 486 with at least four meg of ram. We're talking an investment of over six thousand dollars for a halfway decent multi-tasking operating system and the hardware to run it on. And OS/2 is not multi-user. Even IBM themselves admit that.

It's obvious that we can do better. We CAN one up the PC world. We CAN bring products out on time, we CAN come up with new software without it crashing the customer's machine, and yes, we CAN bring the news and views to you on a regular basis. If we get our act together, we can beat the PC world to the punch. We have the people with the brains - they don't (or at least didn't). We just have to get our collective rear ends into gear and get the job done.

And I'm not just saying all of this to motivate you. I believe it. I've seen it with my own eyes. The difference that good software design can make will beat fancy hardware most every time.

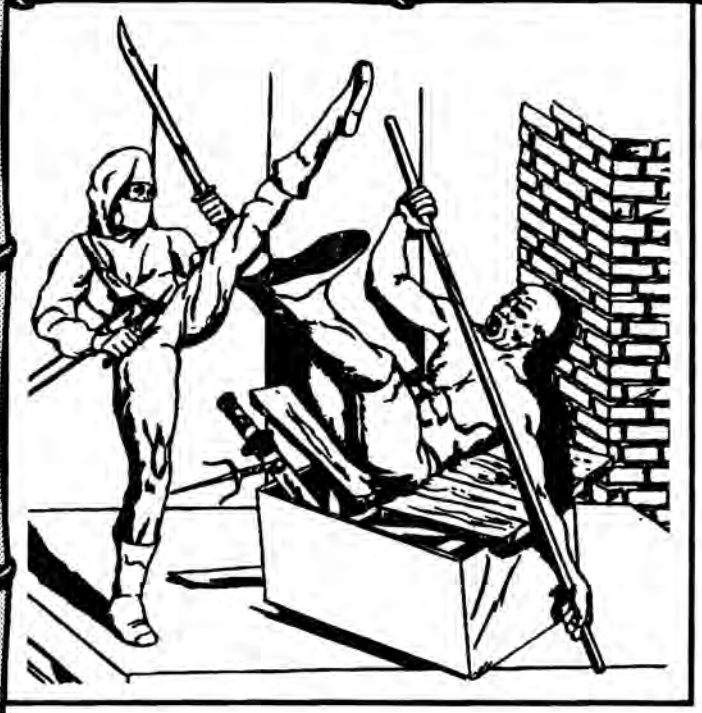
The bottom line is simple. We've got the better operating system, and the better processor. We know it, and they'd know it too if we told them. But we have a choice. To get together and fight for a better future in computing, or do nothing and let the PC's take over. Basically, it's a fight between effeciency or wastefulness.

But more than that, for us it is a fight between existance (to the rest of the world) and extinction. I don't for one minute fool myself into thinking we can take over the world. There are too many PC's out there right now. But we can certainly assure ourselves a place in it's future, by doing the best job we can today.

AND, just to show you that in fact there are people getting the job done out there, I have the following news to report to all of you.

1) The OSKer is back for good.
2) The IMS MM1 machine is going into a production run. Machines are expected to be available at or after the Rainbow Fest.
3) The FHL TC70 is in production and have been shipping since January.
4) The FHL TC9 is undergoing test production runs, some test units sold. Ken-Ton is to have a version of RGB DOS for it.
5) FHL is working on a portable version of the TC70.
6) FHL will be selling inexpensive 68k co-processor boards for TC9.
7) The next OSKer will feature a complete review of the MM1.
8) The issue after will feature a complete review of the TC70.
9) Submissions (programs, articles, hate mail, suggestions) welcome!

See ya next month!



KYUM-GAI TO BE NINJA

Kyum-Gai: to be Ninja (OS-9 Version) is the culmination of a project started almost a year ago. The talents of *Glen R. Dahlgren* (RS-DOS game writer for Sundog Systems), *Kevin Darling* (a legend for his work in OS-9), and *Eddie Kuns* (author of KBCom) have been pooled to create a masterpiece of game software under the OS-9 operating system. Fast martial arts action with outstanding graphics, great digitized sound effects, and incredible animation are featured in this arcade game, all in the OS-9 environment.

Always wanted to play the great CoCo 3 games but didn't want to sacrifice your OS-9 features? ***Multitask while playing*** Kyum-Gai. ***Have multiple Kyum-Gai's*** running in memory. Don't worry about switching windows, because Kyum-Gai: OS-9 auto-pauses, to wait for your return.

Put simply, this is an unprecedented piece of software for the CoCo; a landmark game sure to be a major part of the Color Computer history. Don't miss out on this game!
Req. 512K CoCo III with OS-9 Level 2 and joystick.

$29.95

VISA, Mastercard, Money Order, and COD (USA only, please) accepted. All foreign orders must be sent in US currency Money Orders. Include $2.50 for shipping in USA and Canada, $5.00 Foreign. $3.00 extra for COD orders. VA residents please add 6% sales tax. Dealer inquiries welcome. Authors: we're looking for new software!

Sundog systems

P.O. Box 766
Manassas, VA 22111
(703) 330-8989

O_Say_Can_You_C

Installment #2
by Al Pleagle

Well, its time for the next installment of O_Say_Can_You_C. I hope everyone got their C compilers up and running with the first installment. So lets get busy.

Lets start where a C program should start, with 'main()'. This is where a C program begins execution. If there's no 'main()' the compiler won't know where to start. So if you want an error meltdown, try compiling a C program without it. And more than one 'main()' will confuse things too, although I haven't tried it. So let's write a very short C program. From the OS9 prompt type the following:

build main.c

You will see a '?' at the start of each line. Type each response as indicated. Press the enter key whenever you see <enter>.

Prompt (Do not type this)	Response (Type this)
?	main()<enter>
?	{<enter>
?	}<enter>
?	<enter>

Type:

list main.c

You should see:

```
main()
{
}
```

We know 'main()' is where the program starts, but what about those '{' and '}' brackets? Those brackets mark the beginning and end of 'main()'. The function 'main()' contains whatever is between the '{' and the '}'.

Compile the program by typing:

```
cc1 main.c
```

After the compiler has finished you should be able to call up a directory of /dd/CMDS and find a program called 'main'. At the OS9 prompt, type

main

and press enter. The next thing you will see, surprise, surprise, is the OS9 prompt. The program loaded and executed. Only, it did absolutely nothing, just as it was designed to do. (No, I haven't lost all my marbles, yet.)

Now lets do a little exploration. At the OS9 prompt type the following:

```
ident /dd/cmds/main
```

You should see the following:

```
Header for:  main
Module size: $01A2    #418
Module CRC:  $2B7DC4 (Good)
Hdr parity:  $8B
Exec. off:   $001B    #27
Data Size:   $03D1    #977
Edition:     $01      #1
Ty/La At/Rv: $11 $81
Prog mod, 6809 obj, re-en, R/O
```

Take a close look at what this tells you. Main, which does absolutely nothing and is the simplest possible program in C, is 418 bytes long and requires 977 bytes of data storage. That's over 1K of memory just to get the C compiler to generate a program, any program. So C is obviously not as efficient as assembly language. And its not as easy to understand as Basic09. So why bother with it? Because C is the most portable language for personal computers today. An example is a recently released message editor written in C on a Radio Shack Model III and ported to the Color Computer. Now, no offense intended, but there aren't many computers as dead as the Model III. Maybe the TI-99/4A, its deader 'n hell, but even on a "dead" computer C still gives you the ability to be on the cutting edge of software development.

I'm no longer going to explain step-by-step how to use the 'build' command to enter your source code. If you have a word processor that you like (mine is Dynastar), use it. Type in the listings exactly as printed, save them to disk (be sure to name them with a '.c' at the end) and compile them exactly as we've done earlier.

Let's start something useful. Since almost everyone is familiar with RS-Dos, we'll write a program to clear the screen just like 'CLS' under RS-Dos. We'll call it 'cls' because the convention in OS9 is to capitalize only the names of directories. Type the following source code into a file named 'cls.c':

```
#include <stdio.h>

main()
{
    putchar('\014');
}
```

Next, compile the source code by typing:

```
cc1 cls.c
```

Now type 'ident /dd/cmds/cls' and this is what you should see:

```
Header for:  cls
Module size: $0797    #1943
Module CRC:  $D97896 (Good)
Hdr parity:  $B8
Exec. off:   $001A    #26
Data Size:   $04A3    #1187
Edition:     $01      #1
Ty/La At/Rv: $11 $81
```

```
Prog mod, 6809 obj, re-en, R/O
```

That's 1943 bytes for the program and 1187 bytes for data storage. Pretty hefty for a program that only clears the screen. Just for comparison, look at an assembly language version written by Eddie Gilmore.

```
Header for:  CLS
Module size: $0029    #41
Module CRC:  $F57D96 (Good)
Hdr parity:  $01
Exec. off:   $0012    #18
Data Size:   $00D0    #208
Edition:     $12      #18
Ty/La At/Rv: $11 $81
Prog mod, 6809 obj, re-en, R/O
```

Only 41 bytes for the program, that's over 1900 less than the C version. Data storage only takes 208 bytes, almost 1000 less than the C version. The total for the assembly language version is less than one-tenth the size of the C version. Why do I point this out? To help explain something in our C source code. Lets look at the source code again.

```
#include <stdio.h>

main()
{
    putchar('\014');
}
```

That first line, '#include <stdio.h>', I haven't explained that yet, have I? Well, that is known as a preprocessor directive. It tells the compiler to go find a file named 'stdio.h' and put whatever is in that file into the source code. Inside 'stdio.h' is information the compiler needs for input and output. If you want to take a look, type the following:

```
list /dd/defs/stdio.h
```

The name 'stdio.h' is shorthand for 'standard input output headers' and contains information to handle all types of input and output routines. Since the compiler doesn't know which of those routines it'll need, it includes them all. And that is where a lot of the overhead or extra bytes come from. However, once included, the routines can be called as many times as necessary without any further increase in program size. So while the overhead is excessive on a small program like 'cls', on a larger program like Dynastar, the overhead becomes insignificant. Ease of programming and speed of development become much more important than a couple of thousand bytes. If you're not going to do any input or output, you'll never need 'stdio.h'. But that pretty much limits us to the 'main.c' program we've already written. And I don't think we need any more programs that do absolutely nothing.

Lets look at the other line of code that wasn't included in 'main.c'.

```
    putchar('\014');
```

This is where the program does the real work. This line calls a function 'putchar()'. (Note that 'putchar()' is not the same as 'Putchar()'. C is case sensitive, unlike OS9.) 'putchar()' is a function, just like 'main()'. You can tell they are both functions as their names are followed by two parentheses, '(' and ')'. But you will notice that there is something between these parentheses in the 'putchar()' function in our source code. This is known as the argument. The argument is the value that the function 'putchar()' is to use. In our case the argument is '\014'. 'putchar()' takes that argument '\014' and PUTs the CHARacter out to the terminal. We will discuss the meaning of '\014' shortly.

'putchar()' is contained in the C library as are many other functions which we will discuss in the future. Right now it is important only that you know that C is a language of functions, the more functions in your library, the easier it will be to perform complex tasks in C.

Now let's look at that argument '\014'. C interprets this to be an octal number. Octal simply means base eight. If you were counting in base eight, it would go, "Zero, one, two, three, four, five, six, seven, one-zero." Even though that one-zero would be printed as '10', it is not ten. It is one-zero, base eight, which equals eight in the decimal system with which we are all familiar. (I'm talkin' fingers and toes numbers to the rest of you people from Arkansas.)

So, what does this '\014' mean? To figure that out, count from the right end of the argument toward the backslash (\) starting with zero. Zero, one, two digits are shown. Take the first number on the right and multiply it times eight to the zero power. (Zero was its number in the count from right to left.) So we have four times eight to the zero power. Any number to the zero power is one, so we have four times one, or four. Next take the second number from the right and multiply it times eight to the one power. (One was its number in the count from right to left.) Eight to the one power is eight, so one times eight to the one power is one times eight, or eight. Finally, take the third number from the right and multiply it times eight to the power of two. (Two was its number in the count from right to left starting with zero.) We have zero time eight to the power of two, or zero times sixty-four. Zero times anything is zero, so we have zero. Now, add the values together. Zero plus eight plus four. That equals twelve. So the argument passed to the 'putchar()' function is equal to twelve.

```
        2    1    0 <--Count

        8X8  8    1         <--Powers of 8

    \   0    1    4         <--Base 8 number (indicated by '\')

        64X0 8X1  1X4       <--Multiply number by powers of 8

        0    8    4         <--Results of multiplication

        0 +  8 +  4 =  12   <--Decimal equivalent of '\014'
```

Twelve is the ASCII code to clear the screen on the Color Computer, so when 'putchar()' writes the value twelve to the terminal, the screen is cleared. Don't ask me why twelve does the trick and not thirteen, somebody somewhere just liked twelve, I guess.

There are other ways to set the argument for 'putchar()' equal to twelve. If you have the C Compiler manual handy, look at page 1-4.

Under the heading "Control Character Escape Sequences" you will find the following information:

```
bit patterns:  \NNN     (octal constant)
               \dNNN    (decimal constant)
               \xNN     (hexidecimal constant)
```

So the line with twelve expressed as an octal number,

```
putchar('\014');
```

can also be written with twelve expressed as a decimal number:

```
putchar('\d012');
```

or with twelve expressed as a hexidecimal number:

```
putchar('\x0c');
```

Try substituting these lines and re-compiling the cls.c source code.

One last thing to notice about the line,

```
putchar('\014');
```

There is a semicolon at the end of the line. This tells the C compiler that here is the end of a statement. Those of us who are used to programming in Basic09 or other basic languages generally end the line with just a carriage return (pressing ENTER). That doesn't work for C. You must have the semicolon to tell where one statement ends and the next begins. I'm sure we'll forget a bunch of 'em before we become proficient at C.

Now, lets add some comments to our source code.

```
#include <stdio.h>      /* Tells the compiler to include */
                        /* standard input output headers */

main()                  /* Tells the compiler here is */
                        /* the start of the program   */

{                       /* Here is the start of */
                        /* the function main()  */

    putchar('\014');    /* Put the value 12    */
                        /* out to the terminal */

}                       /* Here is the end of  */
                        /* the function main() */
```

As you can see, comments can be added throughout the source code. They must begin with '/*' and end with '*/'. The compiler knows to ignore anything between those symbols. Although I am the worst about commenting my source code, do as I say, not as I do. Comments can save you untold headaches. Take a break and go try to read some uncommented Basic09 code you've written months ago or, even worse, uncommented code someone else has written. Remember, Basic09 code is much more readable than C. So if you have trouble with that, think how much trouble you'll have with uncommented C code. Do yourself a favor, comment your code.

This is all the further we will go this time. I know many of the advanced C programmers find little value in this article. However, I will ask for their help. There are many different libraries, compilers, header files, configurations and ways to hold your tongue. (Sometimes it won't work unless you hold your tongue just right.) I would ask those of you who program in C on the Color Computer to drop me a line and tell me what you use, which library, whose C compiler, do you compile using a ramdisk, etc. Here is the chance to voice your opinion. If you think you have the best system for compiling C programs on the Color Computer, let me hear about it. And I need as much detail as you care to give me. I will be attempting to determine the 'de facto' standard for C programming on the Color Computer, and I intend to work to that standard. So here's your chance to vote that you do C right.

Until next time, keep smilin'. It makes people wonder what you've been up to.

Al Fleagle
11 Alpine Court
Little Rock, AR 72205

501/661-1063 (voice)

501/661-0527 (sysop@WorkShop)

CIS 72527,1354

Using System Calls
by Paul Pollock

In all things worthwhile, we often find ourselves with a problem that cannot be solved in a conventional fashion. Programming in Basic09 is no exception! Anyhow, this article hopes to help you beginning Basic09 programmers, learn a method to find extra tools from within OS9 itself, to solve difficulties.

To make things easier, we'll examine a couple of common real-life problems (as examples), and solve them with practical solutions; only solvable with a 'System Call'. These solutions will be demonstrated by Basic09 programs which actually use OS9 to cure itself.

Example #1 - Performance Improvement

We've all complained about it. When we execute more than one program in multitask, quite often a Basic09 program slows down the system for another program. While this happens with other types of programs, Basic09 programs (using conventional tools within the language) tend to reduce a systems' thruput much more severely.

While ASSEMBLY (and other forms) solves this problem in clever ways; the question is, how do we apply such solutions to our Basic09 programs? For a clue, let's examine Basic09 itself.

Firstly, let's recognize a simple fact;

The computer waits for input, most of the time.

Because of this fact, Basic09 has been written to take this into account. In all the keyboard entry points; System Mode, Edit Mode and Debug mode; Basic09 is written to operate only long enough to scan the keyboard, and then return to the OS9 polling table as soon as it can. You can test this yourself, by operating another program after starting Basic09. You'll find Basic09 has little effect on the rest of the system.

Except when it is doing something other than keyboard entry; like 'pack'ing a program. This is because Basic09 was written to generate 'pack'ed modules in an efficient manner; and it is necesary to do this (and some other tasks), as fast as it can. To get maximum performance, it now uses it's entire polling clock period, instead of delaying its operation. All of a sudden the computer runs quite slugishly, while packing a large program.

Now to make use of one method to get this technique into our own programs. Let's take a look at the following listing;

```
PROCEDURE Timer
0000    (* Timer program to use the f$sleep System Call
0037    (* Programmed by
0047    (* Paul B.Pollock
0058    (* 8700 Parthenia Place #5
0071    (* Sepulveda, CA 91343
0087    PARAM TIMEOUT:BYTE
008E    TYPE REGISTERS=CC,A,B,DP:BYTE; X,Y,U:INTEGER
00B3    DIM REGS:REGISTERS
00BC    DIM PATH,CALLCODE:BYTE
00C7    REGS.X=TIMEOUT
00D3    CALLCODE=$0A
00DB    RUN SYSCALL(CALLCODE,REGS)
00EA    END
```

Firstly, look at how the program is written. The program generates a TYPE statement (DIM'd as REGS) to tell Basic09 what the 6809 CPU looks like. Then it sets up the registers of the CPU for a System Call.

Note the CALLCODE. This is a mandatory parameter (in this case $0A, used to indicate the F$Sleep call, page 8-35 of the Technical Reference), which must be used to call the SYSCALL command from Basic09.

SYSCALL (included in your CMDS directory), uses the CALLCODE to interface to the System Call table. The rest of the data is sent via a packet defined by the REGS data packet. The only register we are modifying, is the 'X' register, which the Technical References section of the manual tells us is used to tell the System Call how long to TIMEOUT the calling process. All the registers must be sent, via the TYPE packet, but any that are undefined, are assumed to be "don't care' or 'leave as is' info.

Here's the rules for use of the F$Sleep System Call, as used by TIMER. Call the program via the following line;

RUN Timer(TIMEOUT)

The parameter TIMEOUT is an integer number, which determines the number of system 'ticks' you wish the F$Sleep period to be. F$Sleep is NOT repetitive, so you have to repeat this call, everytime you do an input check.

The TIMEOUT can be any number between '0' and '255'.

If '0' is used, then the F$Sleep call will be forever. It has the effect of using LOOP without an EXITIF. The only way to exit is to setup a software interrupt, through a system intercept. This is very complex, and not within the scope of this article.

If '1' is used, the effect of this call is to release any unused time left in the Basic09 program's (which calls Timer) polling interrupt; back to the system, for use by another program.

Any other number used (within range), will cause the F$Sleep to operate for increasingly longer periods. On a standard Coco Level-2 system, these ticks are actually 100/second (even though the manual states 60/second). This means you would normally use 100 ticks for a 1 second TIMEOUT.

The above program assumes that your program has a place in the program where input is expected from the user. For instance, you could make use of this program, right after an INKEY entry point. While the effect of this routine will be noticable with a TIMEOUT of '1', a keyboard scan could get away with a TIMEOUT as large as 2-3 without noticable effect in the program. And would provide an even more dramatic smoothing of system performance. This works well 'because' we are using a System Call. This has no effect on the input drivers and buffers, used to hold data inputted. Characters will be grabbed and stored until called for by the program. If this kind of stoppage were tried with conventional methods, the program would miss characters, during TIMEOUT periods.

Or it could be used by a BBS program, to suspend the module, used to check for a 'carrier detect', which would indicate a user is present. With no user present, the computer does not need to scan for this condition on a continuous basis, because the carrier from a caller is a 'steady-state' condition, and will be there when you eventually get around to it <grin>. This means that you could put the process to sleep for a relatively long period, without affecting the main programs' ability to start the BBS. You could use a period up to one(1) second or more, without the caller being aware of a problem. It could also be used at points in a Menu Control routine, while a user is online. This would prevent long periods of user inactivity from having an adverse effect on your use of the computer. And can be handled as you would any other input scan.

Example #2 - Creating Sound

Those of us who've operated our Coco2's and Coco3's under Extended Color Basic, remember the powerful and elegant methods created by the people at Microsoft; to make not only noises, but sound and music. Commands like SOUND and PLAY, are not only not part of Basic09; but are sorely missed by many programmers.

To alleviate this much loved feature, at least to some extent; let's look at the following program;

```
PROCEDURE Tone
0000    (* Tone Generator Program
0019    (* by Paul Pollock
002B    (* DELPHI <PAULBELL>
003F    (* phone# (818)895-1966
0056    (*
0059    (* This  program is intended as a general purpose Tone
          source.
0096    (* Makes  use of, and demonstrates the use of I$setstt
          System Call.
00D8    (* In  this example, Syscall is used to call  F$SetSta
          (callcode $8E).
011A    (* The actual SS.Tone is option $98.
013E    (* The fine details are in your manual.
0165    (*
0168    (* Usage:
0172    (* RUN Tone(Volume,Tick,Frequency)
0194    (*
0197    (* Params: (all are mandatory)
01B6    (* Volume=   0-63   63 is loud
01D5    (* Tick=     0-255  255 is long (100 ticks/sec)
02F4    (* Frequency=0-4095 4095 is highest pitch
022E    (*
0231    TYPE registers=CC,A,B,DP:BYTE; X,Y,U:INTEGER
0256    DIM regs:registers
025F    DIM path,callcode:BYTE
026A    PARAM Vol,Tick,Freq:INTEGER
0279    DIM xdata:INTEGER
0280    path=0
0287    xdata=Vol*256+Tick
0297    callcode=$8E
029F    regs.A=path
02AB    regs.B=$98
02B7    regs.X=xdata
02C3    regs.Y=Freq
02CF    RUN syscall(callcode,regs)
02DE    END
02E0
```

Like the previous example, this program models the 6809 CPU, and sets this up though the REGS data packet. And, like the previous example, this program recieves several parameters from standard input. These parameters are used to modify the CPU registers as listed. The CALLCODE ($8E), is the key for the F$Setsta call. This call is the doorway for a 'sub-table' of additional system tools. These tools become available via the 'B' register of the CPU. This register contains the tag ($98) for 'SS.Tone'. The 'X' register contains a 2-byte code, which contains the volume and duration data. The 'Y' register contains the Frequency information.

All that's left is to send this data to SYSCALL, and let it rip! The limits are stated in the program, and decimal integers are used for all parameters. This program can be used to generate tones up to 2.5 seconds long <grin>. This program is deceptively safe, as screwing up the parameters will have little effect on the system. You might not get a tone, but the computer will continue to operate normally.

One other important feature of this program is; unlike some programmer's other methods for sound production, this routine will NOT send random data to the printer. For this reason, it is ideal for cases where you wish to make sounds and still use your printer.

The only bad part of this program, isn't really part of this program, but a part of OS9. Since OS9 is a multitasking operating system, the system wants to run its programs while the program wants to make a tone. This program does not halt the system to run. So the effect of the system on this program is that it makes the sound seem 'grainy' or 'buzzy'. This effect is mildly unpleasant, but is otherwise quite effective.

Wrapping Up The Loose Ends

I hope this article has been informative, and fun for you. The programs included in this article are yours to use as you wish, as I've released them into the public domain.

The Basic09 programs, 'Timer' and 'Tone', work on OS9 Level-1 and OS9 Level-2 systems without modification (OSK systems, confirm the system callcodes used, and any other parameters required). They pack into very small procedures, and require very little data, so they lend themselves to larger projects where you might like to include them into merged Basic09 procedures. The only external module required under Basic09 is SYSCALL. Packed procedures will also require RUNB, for proper operation.

Good Luck, and keep programming!

Z_End

That's all for today's issue folks!

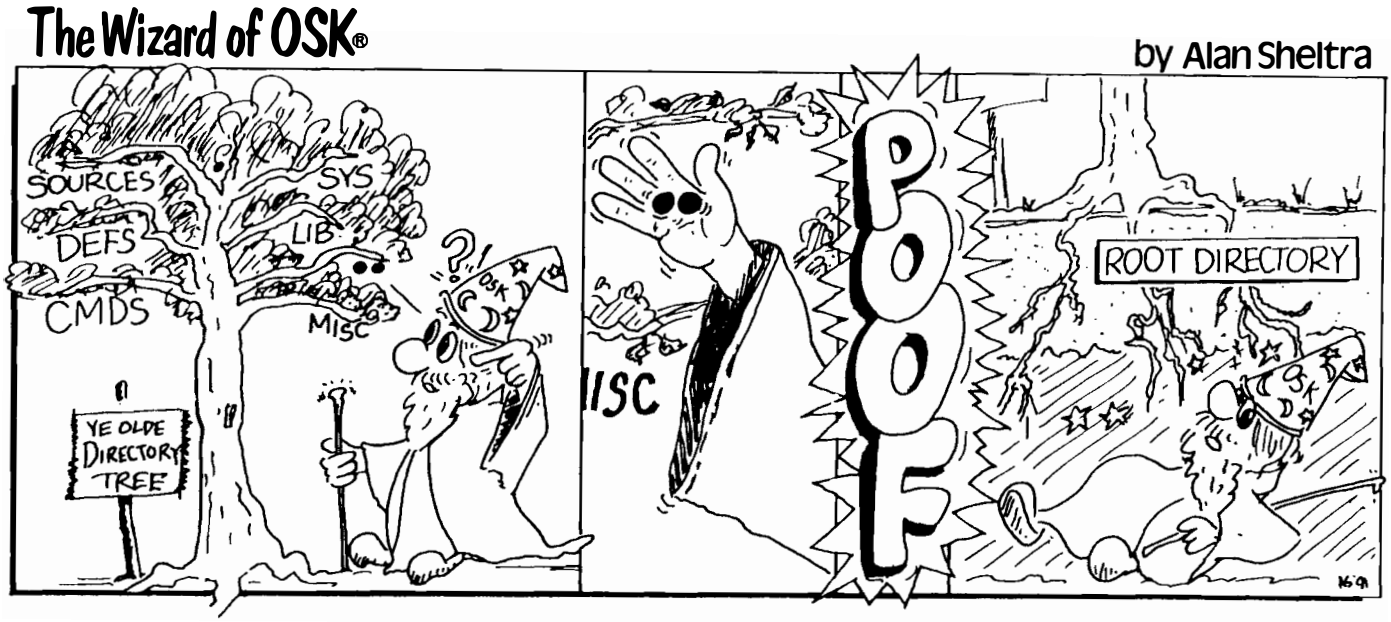
Look forward to our following issues reviewing the various machines either now or soon to be available. We start off with a complete top down review of the MM1 from the inside out. That's right, everything you didn't want to know about the MM1!

And remember, Please mention the OSKer when you contact our advertisers. It never hurts to let them know which magazine you found them in!

Does it sound like I'm stalling to you? (I needed just one more page.. not two or three or four...).

Also in next months issue: the next installment of Playing Chess in C, and also Bug's Windows, Bugs, and Patches article. That's right, we call the guy Bug.

See ya next month!



SYSTEM IV COMPUTER

PERFORMANCE - FLEXIBILITY - VERSATILITY

MC68000 Microprocessor - 16 MHz
1 MBytes of DRAM (0 wait state)
4 MBytes optional

VGA Video Card - 800 x 600 x 16
to 320 x 200 x 256
or - Hercules Monochrome Card
Seven IBM/XT Compatible slots
Floppy Disk Controller (37C65)
supports two 360K, 720K, 1.2M
or 1.4M Drives
One 1.4 MByte, 3 1/2" Floppy Drive
WD XTGEN Hard Disk Controller (MFM)
Four RS-232 Serial Ports
Parallel Printer Port
40 MB Hard Drive, 28 msec, optional
Clock with Battery
4 layer board
Memory Expansion slot - adds
up to 8 MB of DRAM.
AT style keyboard
200 Watt Switching Power Supply
Professional OS9/68000
Drivers and descriptors for the
devices and ports provided.
Baby AT Case - holds three 5 1/4"
1/2 ht drives and one 3 1/2"
drive accessable from the
front and one 3 1/2" internal
drive.
One year parts and labor warranty
Mfg by Peripheral Technology

Model K402-m with Hercules Monochrome Video Card	$1,399.00
Model K402-v with VGA Video Card	$1,499.00
Model K401-m with 4 MB on-board DRAM, 40 MB Hard Drive (28 msec) and Hercules Monochrome Card	CALL FOR NEW LOW PRICES
Model K401-v with 4 MB on-board DRAM, 40 MB Hard Drive (28 msec) and VGA Card	CALL FOR NEW LOW PRICES
14" VGA Monitor, 0.31 mm, Gold Star 1450 Plus with K401-v or K402-v (Regular $549.00)	CALL FOR NEW LOW PRICES
14" Monochrome TTL Monitor, Amber, Gold Star 1401A with K401-m or K402-m (Regular $199.95)	CALL FOR NEW LOW PRICES
VGA Monochrome Monitor, white, Tandy VGM-100 with K401-v or K402-v (Regular $199.95)	$ 165.00

OS9/680x0 SOFTWARE

SCULPTOR - Development System (68000) from	$2,500.00
Quick Ed - Editor and Text Formatter	$ 275.00
FlexiLint - A must for C programmers	$ 495.00
Caching - High speed disk caching (demo available)	$ 300.00
IMP - Intelligent Make Program	$ 250.00
Disassembler - 3 pass	$ 250.00
Windows - C source code & library	$ 250.00
Profile - Tune User State Programs	$ 270.00
PAN Utilities - C source and library	$ 250.00

delmar co

P.O. BOX 78 • MIDDLETOWN SHOPPING CENTER MIDDLETOWN, DE 19709 302/378-2555